Theoretical Computer Science 2 (1976) 225–247.

RELATIONSHIPS BETWEEN CLASSES OF MONOTONIC FUNCTIONS*

Mark B. TRAKHTENBROT
Computing Center, Novosibirsk 630090, USSR

Communicated by A. Ershov
Received July 1975

Abstract. The class of monotonic (in the sense of Scott) functions is divided into subclasses of sequential and parallel ones and the problem of comparative power of different sequential and parallel functions with respect to composition (Theorems 1–5) and recursion (Theorem 6) is then investigated. In particular, Theorem 6 answers the question of Scott concerning the power of his Logic for Computable Functions.

0. Introduction

One of the key notions in mathematical theory of computation is that of monotonic function [6]. Some of these functions are of special interest in programming languages, e.g. the conditional *if-then-else*. In particular, using this function allows recursive definitions to be represented in a convenient manner suggested by McCarthy [2].

In studying monotonic functions an essential point is classifying them as sequential or parallel. The reason for such classification is that, in general, before computing a value of a function, its arguments should be computed and that computation of an undefined value never terminates. So, an order (or a strategy) of arguments computation is of particular importance. This fact has been already pointed out in earlier papers on the theory of computation. For example, McCarthy [2] noted it when describing a system of equivalent transformations of conditional expressions.

Informally, a function is sequential if its arguments can be computed in a sequence and the values already computed determine which of the remaining arguments is the next to be computed. If such a strategy does not exist then a function is parallel. For example, the *if-then-else* function is sequential. The formal definition of sequentiality was given in [8]. It was also shown in that paper that no parallel function can be derived from sequential ones using usual programming features such as composition and recursion. The problem of increasing the power of

* This is an extended version of a paper originally presented at the Symposium on Mathematical Foundations of Computer Science, Mariánské Lázně, Czechoslovakia, September 1975.

languages by adding parallel functions has been studied by a number of authors. For example, it follows from [8] that only sequential functions (but not all — see below) can be expressed in Scott's *logic for computable functions* (LCF) [5]; in [5] Scott asks whether an addition of the parallel disjunction *OR* to LCF is reasonable. In [3] Paterson and Hewitt introduced the *IF-then-else* function, which is a parallel variant of *if-then-else*. They have established that the use of *IF-then-else* allows us to increase the power of recursive schemata.

In [7] the author defined another parallel function Γ (the voting function) which allows us to define a still more powerful class of recursive schemata. Moreover, this class is, in a sense, of maximal power. This maximality follows from the existence of a finite basis (with respect to composition) in one special class of monotonic functions and also from the form of this basis.

In the present paper we investigate in detail the problem of comparative power of certain sequential and parallel functions with respect to composition (Sections 3 and 4, Theorems 1–5) and recursion (Section 5, Theorem 6). In fact Theorems 3 to 6 establish the completeness of some rather simple constructions (such as *if-then-else*, *IF-then-else*, *OR*, Γ) in different classes of sequential and parallel functions. Theorems 3 and 4 allow us also to establish the difference between two concepts of effectiveness: computability (when, given the values of arguments, the value of a function is effectively obtained) and effective sequentiality (when there exists an effective strategy of the argument computation). Namely, it is proved that computable sequential functions exist which are not effectively sequential. Back again to the problem of LCF power we should like to mention paper [4] in which it was proved that effectively sequential functions and only they can be expressed in LCF. Answering the question of Scott, our Theorem 6 shows that adding the *OR* function to LCF allows one to express all computable monotonic functions. Thus the results obtained define more precisely the role of *if-then-else* and the related constructions in the framework of algorithmic languages oriented to a description of sequential and parallel functions.

All proofs in this paper are given in separate sections. Theorems 1, 2 and the part of Theorem 4 which contain negative results are proved in Section 7, while Theorems 3, 5 and the second part of Theorem 4 are proved in Section 9. Auxiliary notions and lemmas necessary for these proofs are given in Sections 6 and 8 accordingly. Routine details are, as a rule, omitted.

1.

Let us recall some necessary notions (see, e.g., Scott [6] and Manna et. al. [1]). Let D be an arbitrary data domain containing a so-called undefined element ω; $B = \{true, false, \omega\}$ (we use the same denotation for an undefined boolean and an undefined data element). We introduce the partial ordering $\subseteq$ on $B^k \times D^m$ $(k, m \geqslant 0; k + m = n)$ meaning "is less defined than or equal to":

(a) for any $x \in D$ ($x \in B$) let $\omega \subseteq x$ and $x \subseteq x$, and for any $y \in D$ ($y \in B$) which is identical with neither x nor ω let $x \not\subseteq y$ and $y \not\subseteq x$;

(b) for vectors $\bar{x}$ and $\bar{y}$ with components $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ let $\bar{x} \subseteq \bar{y}$ when $x_i \subseteq y_i$ for each i.

A function $f(\bar{x})$ mapping $B^k \times D^m$ into D (or, in other words, a function of the *type* $b, d \to d$) or into B (a function of the *type* $b, d \to b$) is *monotonic* if for all $\bar{x}$ and $\bar{y}$, $\bar{x} \subseteq \bar{y}$ implies $f(\bar{x}) \subseteq f(\bar{y})$. For two monotonic functions $f(\bar{x})$ and $g(\bar{x})$ we write $f \subseteq g$ if $f(\bar{x}) \subseteq g(\bar{x})$ for all $\bar{x}$.

Some important examples of monotonic functions are given below.

(1) Let f be an arbitrary partial function from $(B \setminus \{\omega\})^k \times (D \setminus \{\omega\})^m$ into $D \setminus \{\omega\}$ or into $B \setminus \{\omega\}$. Then we define its *natural extension* $\tilde{f}$: if $f(\bar{x})$ is defined and equals y, then $\tilde{f}(\bar{x})$ also equals y; otherwise (in particular, whenever at least one of its arguments is ω) $\tilde{f}(\bar{x})$ yields the value ω. Clearly, f is monotonic.

(2) We denote the natural extension of the equality predicate by $x = y$ to distinguish it from the other equality, $x \equiv y$; the latter one means that $x \subseteq y$ and $y \subseteq x$ and is clearly nonmonotonic. We write $f \equiv g$ if $f \subseteq g$ and $g \subseteq f$.

(3) The *OR* function is an extension (not the natural one) of the usual disjunction by letting ω *OR true* $\equiv$ *true* and *true OR* $\omega \equiv$ *true*. The *or* function differs from *OR* in that ω *or true* $\equiv \omega$.

(4) Two conditional functions, *if* α *then* x *else* y and *IF* α *then* x *else* y (further denoted by *if* and *IF*), map $B \times D^2$ into D. They differ in that for all x and y *if* ω *then* x *else* $y \equiv \omega$, but *IF* ω *then* x *else* $x \equiv x$. Clearly, *if* $\subseteq$ *IF*.

(5) The voting function Γ (introduced by Trakhtenbrot [7]) mapping D^3 into D is defined by

$$\Gamma(x, y, z) \equiv \begin{cases} x & \text{if } x \equiv y \quad \text{or} \quad x \equiv z, \\ y & \text{if } y \equiv z, \\ \omega & \text{otherwise.} \end{cases}$$

In a similar way we can define boolean variants of the functions *if*, *IF* and Γ having the type $b \to b$. To distinguish between these two variants we will use denotations if_t, IF_d, Γ_d and if_b, IF_b, Γ_b respectively.

(6) A constant function $f(x_1, \ldots, x_n)$ is defined by letting $f(\bar{x}) \equiv c$ for all $\bar{x}$. This means that either f is always undefined, i.e. $c \equiv \omega$ or $f(\omega, \ldots, \omega) \equiv c \not\equiv \omega$. A nullary constant function will be referred to as a constant.

A monotonic function $f(\bar{x})$ is *computable* if there exists an algorithm such that given the values of arguments $\bar{x}^0$ (in particular, some of them may equal ω), it stops and yields the correct result if $f(\bar{x}^0) \not\equiv \omega$ and runs infinitely otherwise. (Of course, it is assumed here that the domain D is a constructive set. From this point on, when considering computable functions, we put $D = \{\omega, 0, 1, 2, \ldots\}$.)

For example, any partial recursive function is extended in the natural way to a computable monotonic function. It is also clear that all monotonic functions of Examples 2–6 above are computable.

2.

The further study of monotonic functions is based on classifying them as sequential and parallel. The reason for such a classification is that in real situations before computing a value of a monotonic function $f(\bar{x})$ one should compute its arguments, and that computation of the value undefined never terminates.

An argument x_i is *critical* if the following condition is satisfied

$$x_i \equiv \omega \to f(\bar{x}) \equiv \omega.$$

The strategy of computation of the arguments is the most simple when f is a constant function (then no computation is needed at all) or a *strict* one, i.e. each of its arguments is critical. In the latter case the arguments may be computed in any order.

In general a *strategy of sequential computation* of a function f is a totality of functions $\{F_1, \ldots, F_{2^n}\}$ (the number of which equals the number of subsets of the arguments set $\{x_1, \ldots, x_n\}$), satisfying the following conditions:

(1) F_1 is a constant; if f is a constant function then $F_1 \equiv f(\bar{x})$ else F_1 equals the number of some critical argument of f;

(2) for $i \neq 1$, $F_i(x_{k_1}, \ldots, x_{k_s})$ is a strict function such that for every $x^0_{k_1}, \ldots, x^0_{k_s}$ the constant $F_i(x^0_{k_1}, \ldots, x^0_{k_s})$ and the function obtained from f by the indicated fixing the arguments $x_{k_1}, \ldots, x_{k_s}$, satisfy condition (1).

(Obviously, there is some surplus in this definition; since for nonconstant function f an argument with the number F_1 must first be computed, we may omit those F_i's which do not depend on this argument.)

A monotonic function f is *sequential* if there exists a strategy of sequential computation of f; otherwise f is *parallel*. (This definition is equivalent to that given by Vuillemin [8].)

A computable sequential function f is *effectively sequential* if for each i, F_i is computable.

For example, any constant function $f(\bar{x})$ is clearly effectively sequential and for each i, $F_i \equiv f(\bar{x})$. The *or* and *if* functions are also effectively sequential. In particular, the strategy for α_1 *or* α_2 is as follows:

$$F_1 \equiv 1;$$

$$F_2(\alpha_1) \equiv \begin{cases} \omega & \text{if } \alpha_1 \equiv \omega, \\ \textit{true} & \text{if } \alpha_1 \equiv \textit{true}, \\ 2 & \text{if } \alpha_1 \equiv \textit{false}; \end{cases}$$

$$F_3(\alpha_1, \alpha_2) \equiv \begin{cases} \alpha_1 \textit{ or } \alpha_2 & \text{if } \alpha_1 \not\equiv \omega \text{ and } \alpha_2 \not\equiv \omega, \\ \omega & \text{otherwise.} \end{cases}$$

The *OR*, *IF* and Γ functions are parallel.

Given a domain D, we will use the following denotations for different classes of

functions: M — monotonic, SM — sequential, I — constants and strict (considered further as initial). In each of these classes we distinguish the corresponding subclass consisting of computable functions: CM, CSM, CI; the set of effectively sequential functions is denoted by ESM. Lastly by $\mathrm{I}(f_1, \ldots, f_k)$ we denote the class of functions which can be derived from $\mathrm{I} \cup \{f_1, \ldots, f_k\}$ by composition, with its subclass $\mathrm{CI}(f_1, \ldots, f_k)$ being defined accordingly. (Strictly speaking, we ought to write M_D, etc. We omit the index because it never leads to ambiguity.)

As usual, compositions are described by *terms* and we will use the same denotations when speaking about classes of functions or classes of terms describing those functions.

For the sake of brevity, functions taking their values in D (B) and the corresponding terms will be called d-functions and d-terms (b-functions and b-terms).

3.

Theorems 1 and 2 below establish comparative power of the sequential and parallel functions defined in Section 1. Here and further we assume that D contains at least two elements nonidentical with ω. We also use the fact that for every D

$$\textit{if } \alpha \textit{ then } x \textit{ else } y \equiv IF\,\alpha \textit{ then } x \textit{ else } (IF\,\alpha \textit{ then } \omega \textit{ else } y) \tag{1}$$

holds, so that $\mathrm{I}(if, IF) = \mathrm{I}(IF)$.

Theorem 1. (1) *If D is infinite then*

$$\mathrm{I}(if_d, OR) \subsetneq \mathrm{I}(IF_d) \subsetneq \mathrm{I}(if_d, \Gamma_d).$$

(2) *If D is finite then* $\mathrm{I}(if_d, OR) = \mathrm{I}(IF_d) = \mathrm{I}(if_d, \Gamma_d)$ *and moreover* $\mathrm{I}(OR) = \mathrm{I}(IF_d) = \mathrm{I}(\Gamma_d)$.

It follows from this theorem that for any D, $OR \in \mathrm{I}(IF_d)$. It can be shown that for any D, $OR \in \mathrm{I}(\Gamma_d)$ also holds and furthermore, that if D is infinite then $IF_d \notin \mathrm{I}(\Gamma_d)$ though $IF_d \in \mathrm{I}(if_d, \Gamma_d)$.

Theorem 2. (1) *If D is infinite then* $\mathrm{I}(or) \subsetneq \mathrm{I}(if_d)$.

(2) *If D is finite then* $\mathrm{I}(or) = \mathrm{I}(if_d)$.

4.

Consider now the following problem: It is required to express any function in M or in some of its given subclasses as composition of the initial and a small number of

extra sequential and parallel functions. Note that extra functions are essential because the classes I and SM are closed under composition (see Vuillemin [8]).

Theorem 3. *For every D,* $\mathrm{SM} = \mathrm{I}(if_d)$ *and* $\mathrm{ESM} = \mathrm{CI}(if_d)$.

Note that if D is finite then all functions in M are computable. Then it follows from Theorem 3 that in this case $\mathrm{CSM} = \mathrm{CI}(if_d)$ and hence CSM = ESM. In general, however, CSM is wider than ESM, as shows the following theorem.

Theorem 4. *If D is infinite then* $\mathrm{CI}(if_d) \subsetneq \mathrm{CSM} \subsetneq \mathrm{CI}(IF_d)$.

Note that $\mathrm{CI}(IF_d)$ contains not only sequential but also parallel functions (and this is the reason for the second containment in Theorem 4 to be proper). It is interesting, that using IF_d (and even the less powerful function OR) allows one to express any 2-ary function:

Proposition 1. *For every D and for every 2-ary function f,*

$$f \in \mathrm{M} \leftrightarrow f \in \mathrm{I}(if_d, OR) \quad \textit{and} \quad f \in \mathrm{CM} \leftrightarrow f \in \mathrm{CI}(if_d, OR).$$

However it follows from Theorem 1 that IF_d is not powerful enough to express all (even 3-ary) monotonic functions. Theorem 5 below shows that the Γ_d function possesses such a power.

Theorem 5. *For every D,* $\mathrm{M} = \mathrm{I}(if_d, \Gamma_d)$ *and* $\mathrm{CM} = \mathrm{CI}(if_d, \Gamma_d)$.

The following propositions show that in some cases Theorems 3 to 5 may be strengthened.

Proposition 2. (1) *For every D and every b-function f,*

$$f \in \mathrm{SM} \leftrightarrow f \in \mathrm{I}(or) \quad \textit{and} \quad f \in \mathrm{ESM} \leftrightarrow f \in \mathrm{CI}(or).$$

(2) *If D is infinite then there exists a b-function* $f \in \mathrm{CSM}$ *such that* $f \notin \mathrm{CI}(or)$.

(3) *For every D and every b-function f,*

$$f \in \mathrm{M} \leftrightarrow f \in \mathrm{I}(OR) \quad \textit{and} \quad f \in \mathrm{CM} \leftrightarrow f \in \mathrm{CI}(OR).$$

Proposition 3. *If D is finite then for a function f of either type (d-function or b-function),*

(1) $f \in \mathrm{SM} = \mathrm{CSM} = \mathrm{ESM} \leftrightarrow f \in \mathrm{I}(or) = \mathrm{CI}(or)$,

(2) $f \in \mathrm{M} = \mathrm{CM} \leftrightarrow f \in \mathrm{I}(OR) = \mathrm{CI}(OR)$.

5.

Using recursive definitions of functions is one of the most important methods of increasing power of languages. Every such recursive definition is considered to define its least (least defined) fixed point.

Let $D = \{\omega, 0, 1, 2, \ldots\}$. In the *logic for computable functions* (LCF) described in Scott [5] operators of composition and recursion are allowed which are applied to a small set of originally given effectively sequential functions (which we denote by $\mathscr{F}$). $\mathscr{F}$ contains conditional if_d, the natural extensions of the functions $x+1, x-1$ $(0-1 \equiv \omega)$, projection $K(x, y) \equiv x$, predicate $x = 0$ and constants 0, *true*, *false*. It follows from Vuillemin [8] that only computable sequential functions can be expressed in LCF. After the author obtained the results formulated as Theorems 3 and 4 (containment $\mathrm{CI}(if_d) \subsetneq \mathrm{CSM}$) Sazonov [4] established more powerful facts. Namely, he showed that all effectively sequential functions and only they can be expressed in LCF. Thus, computable sequential but not effectively sequential functions cannot be expressed even in LCF.

It is discussed in Scott [5] what other functions could be added to $\mathscr{F}$ and, in particular, whether the addition of the *OR* function is reasonable. This extension may be justified by:

Theorem 6. *Every computable monotonic function can be derived from* $\mathscr{F} \cup \{OR\}$ *by composition and recursion.*

This follows from Theorem 5 and Sazonov's remark (private communication) that Γ_d is the least fixedpoint of the following recursion:

$$f(x, y, z) \equiv \textit{if } (x = 0 \textit{ AND } y = 0) \textit{ OR } (x = 0 \textit{ AND } z = 0) \textit{ OR } (y = 0 \textit{ AND } z = 0)$$
$$\textit{then } 0 \textit{ else } (f(x-1, y-1, z-1)+1).$$

Here $\alpha \textit{ AND } \beta \equiv \neg(\neg\alpha OR \neg\beta)$.

6.

In this section some definitions and statements are given necessary for the proofs of Theorems 1, 2, 4. The matters presented are based on those of [7] where similar situations have been considered.

In the classes of terms $\mathrm{I}(if_d)$, $\mathrm{I}(if_b)$, $\mathrm{I}(IF_d)$ and $\mathrm{I}(IF_b)$ we distinguish special subclasses of so-called *canonical terms* denoted by $\mathrm{I}'(if_d)$, $\mathrm{I}'(if_b)$, $\mathrm{I}'(IF_d)$ and $\mathrm{I}'(IF_b)$ accordingly. Definitions of these subclasses are rather similar and are carried out inductively. Terms constructed of symbols of constants, boolean and data variables and strict functions in the usual way, are considered to be initial.

$I'(if_d)$: (a) if τ is an initial d-term then $\tau \in I'(if_d)$;

(b) let τ_0 be an intial b-term other than constants *true* and *false*, and $\tau_1, \tau_2 \in I'(if_d)$, then $if_d\ \tau_0$ *then* τ_1 *else* $\tau_2 \in I'(if_d)$.

$I'(IF_d)$ is defined similarly with replacing if_d by IF_d.

To define $I'(if_b)$ and $I'(IF_b)$ one ought to replace if_d and IF_d by if_b and IF_b, and initial d-terms in case (a) by initial b-terms.

Thus a structure of any canonical term is described by a binary *termal tree* with branch-nodes which we call *tests* (having *true*- and *false*-*successors*) and leaf-nodes which we call *terminals*. Note that all tests and terminals are initial terms. All tests which are constants *true* and *false* are excluded since they are clearly superfluous.

Lemma 1 (See [7]). (1) *For every d-function f,*

$$f \in I(if_d) \to f \in I'(if_d),$$

$$f \in I(IF_d) \to f \in I'(IF_d).$$

(2) *For every b-function f,*

$$f \in I(if_d) \to f \in I'(if_b),$$

$$f \in I(IF_d) \to f \in I'(IF_b).$$

Proof. The proof is based on closeness of I under composition [8] and also on two transformations:

$$f(IF\ \alpha\ then\ x\ else\ y) \equiv IF\ \alpha\ then\ f(x)\ else\ (IF\ \alpha\ then\ (IF\ x = y\ then\ f(x)\ else\ f(\omega))\ else\ f(y)) \quad (2)$$

for any monotonic function f; if besides that $f(\omega) \equiv \omega$ then (cf. [1])

$$f(if\ \alpha\ then\ x\ else\ y) \equiv if\ \alpha\ then\ f(x)\ else\ f(y) \quad (3)$$

(both transformations generalize to any n-ary f). Note that a change of types of *if* and *IF* is possible here. For example, if f is of the type $d \to b$ then if_d and IF_d are replaced by if_b and IF_b. Details are left to the reader. □

Note that replacing $I(if_d)$ by $I(if_d, if_b)$ and $I(IF_d)$ by $I(IF_d, IF_b)$ in implications of the lemma keeps it true. This follows from containments $if_b \in I(if_d)$ and $IF_b \in I(IF_d)$. (It is sufficient to encode *true* and *false* by two data values which are nonidentical with ω and then to decode resulting values of if_d and IF_d. Details of such an encoding are given in the proof of Theorem 1.) Furthermore, it follows from the lemma and equality (1) that

(a) for d-function f: $f \in I(if_d, IF_b) \to f \in I'(IF_d)$,

(b) for b-function f: $f \in I(if_d, IF_b) \to f \in I'(IF_b)$.

However, careful application of transformations (2) and (3) allows us to give a more exact characteristic of d-functions in terms of the class $\mathrm{I}'(if_d, IF_b)$ defined as follows:

(a) if τ is an initial d-term then $\tau \in \mathrm{I}'(if_d, IF_b)$;

(b) let $\tau_0 \in \mathrm{I}'(IF_b)$, $\tau_1, \tau_2 \in \mathrm{I}'(if_d, IF_b)$; then

$$if_d\ \tau_0\ then\ \tau_1\ else\ \tau_2 \in \mathrm{I}'(if_d, IF_b).$$

Thus tests in a canonical term in $\mathrm{I}'(if_d, IF_b)$ are canonical terms in $\mathrm{I}'(IF_b)$ (and therefore they will be referred to as macrotests). As for terminals, they are initial d-terms — just as in $\mathrm{I}'(if_d)$.

Lemma 2. *For every d-function f,*

$$f \in \mathrm{I}(if_d, IF_b) \rightarrow f \in \mathrm{I}'(if_d, IF_b).$$

Proof. Let τ be a term representing a function $f \in \mathrm{I}(if_d, IF_b)$. Modify τ so that all strict functions which occur in it be either of the type $d \rightarrow b$ or $d \rightarrow d$. This can be done by applying a transformation called *if-decomposition with respect to a boolean argument*: If a function $g(\alpha, \bar{x})$ (α is a boolean argument) is such that $g(\omega, \bar{x}) \equiv \omega$ then

$$g(\alpha, \bar{x}) = if\ \alpha\ then\ g(true, \bar{x})\ else\ g(false, \bar{x}). \tag{4}$$

Note that if_d is used in *if*-decomposition when g is a d-function and if_b when g is a b-function.

Now while there is at least one occurrence of if_d in an argument position of a strict function of the type $d \rightarrow d$, apply transformation (3). It is easy to see that as a result a term will be obtained which differs from a canonical one in $\mathrm{I}'(if_d, IF_b)$ only in that its macrotests are in $\mathrm{I}(if_d, if_b, IF_b)$ but not in $\mathrm{I}'(IF_b)$. Note that all occurrences of if_b in macrotests arise as a result of the *if*-decomposition and all occurrences of if_d are only in argument positions of strict functions of the type $d \rightarrow b$. Replacing if_b by IF_b and if_d by IF_d (cf. (1)) transforms all macrotests to those in $\mathrm{I}(IF_d, IF_b)$. At last, using the equality $\mathrm{I}(IF_d, IF_b) = \mathrm{I}(IF_d)$ (see above) and Lemma 1 (2) yields the desired representation of f. □

By using definitions of the classes $\mathrm{I}'(if_d), \mathrm{I}'(IF_d), \ldots$ we may define in the usual way the corresponding subclasses $\mathrm{CI}'(if_d)$, $\mathrm{CI}'(IF_d)$, etc. Clearly, Lemmas 1 and 2 keep valid if we replace simultaneously $\mathrm{I}(if_d), \ldots$ by $\mathrm{CI}(if_d), \ldots$ and $\mathrm{I}'(if_d), \ldots$ by $\mathrm{CI}'(if_d), \ldots$. This follows immediately from the form of transformations (2)–(4).

Computing subtrees and paths [7]. Let an interpretation J of a term τ in $\mathrm{I}'(IF_d)$ or in $\mathrm{I}'(IF_b)$ be fixed, i.e. the values of all variables in τ be given (functional symbols are assumed to be originally interpreted). Then, following the definition of IF, we determine a *computing subtree* Q_J in the corresponding termal tree Q:

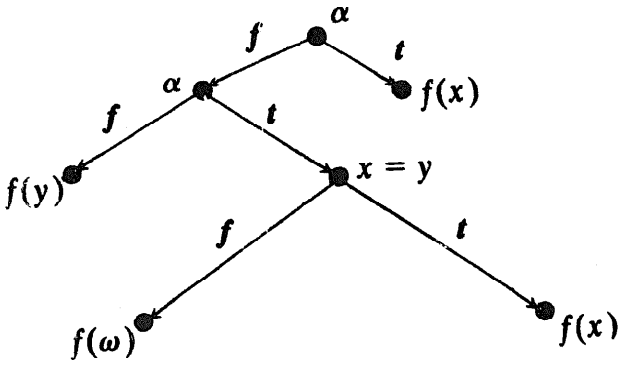
(a) the root of Q is also the root of Q_J;

(b) if a test included in Q_J equals *true*, *false* or ω under J then we include in Q_J its *true*-successor, *false*-successor or both successors respectively.

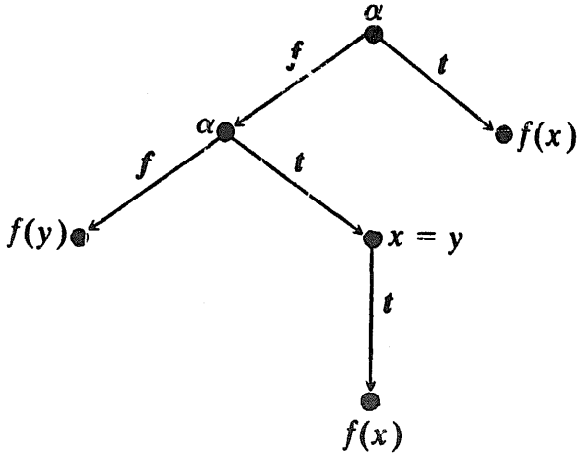
Clearly, if the values of all terminals in Q_J coincide then the whole term τ yields the same value under J; otherwise its value is ω.

If now τ is a term in one of the classes $I'(if_a)$, $I'(if_b)$ or $I'(if_a, IF_b)$ then we determine a *computing path* Q_J in Q, which is a special case of computing subtree. It also is constructed starting from the root of Q. As soon as a test (or a macrotest) which equals ω is firstly reached, this constructing is stopped; following the definition of *if*, the value of τ in this case also equals ω. If such a test does not occur then a path from the root to some terminal will be constructed; its value coincides with that of τ under J.

To illustrate the concepts introduced above we consider the following example. Let in equality (2), f be a strict d-function. Then the right-hand side of this equality is a canonical term in $I'(IF_d)$. The termal tree Q is of the form.



where $x = y$ and both occurrences of α are tests, while $f(y)$, $f(\omega)$ and both occurrences of $f(x)$ are terminals. Let J be such that $\alpha \equiv \omega$ and $x = y$, then the computing subtree Q_J is of the form



Since the values of all terminals in Q_J coincide under such an interpretation (and equal $f(x)$), the whole term yields the same value under J.

7.

In this section we prove Theorems 1 and 2 and the proper containment $CI(if_d) \subsetneq CSM$ (Theorem 4). Here and later on, all positive statements about belonging of a function to a given class are proved "by construction". As for the negative results they are proved by assuming the contrary which is followed by constructing refuting interpretations. In this latter case all terms are assumed to be reduced to the corresponding canonical form (see Lemmas 1 and 2).

Here we assume for simplicity that if D is finite, then $D = \{\omega, 0, 1\}$ and if D is infinite, then $D = \{\omega, 0, 1, 2, \dots\}$.

Proof of Theorem 1. (a) We first show that for every D, $I(if_d, OR) \subseteq I(IF_d) \subseteq I(if_d, \Gamma_d)$. Let us consider the strict "encoding" function

$$f(x) \equiv \begin{cases} true & \text{if } x = 1, \\ false & \text{if } x = 0, \\ \omega & \text{otherwise.} \end{cases}$$

Obviously,

$$\alpha\ OR\ \beta \equiv f(IF_d\ \alpha\ then\ 1\ else\ (IF_d\ \beta\ then\ 1\ else\ 0)) \tag{5}$$

which together with (1) yields the first containment. Consider, further, the following functions:

$$g_1(x, y) \equiv if_d\ x = y\ then\ x\ else\ \omega,$$

$$g_2(\alpha, x, y) \equiv \Gamma_d(if_d(\alpha, x, y), g_1(x, y), x),$$

$$g_3(\alpha, x, y) \equiv \Gamma_d(if_d(\alpha, x, y), g_1(x, y), y).$$

It is easy to verify that

$$IF_d\ \alpha\ then\ x\ else\ y \equiv \Gamma_d(if_d(\alpha, x, y), g_2(\alpha, x, y), g_3(\alpha, x, y))$$

which yields the second containment.

Let now D be finite. Then, obviously,

$$\begin{aligned} \Gamma_d(x, y, z) \equiv if_d\ & (x = 0\ AND\ y = 0)\ OR\ (x = 0\ AND\ z = 0)\ OR \\ & (y = 0\ AND\ z = 0) \\ & then\ \ 0 \\ & else\ (if_d(x = 1\ AND\ y = 1)\ OR\ (x = 1\ AND\ z = 1)\ OR \\ & (y = 1\ AND\ z = 1) \\ & then\ 1\ else\ \omega). \end{aligned}$$

Hence, in this case $I(\Gamma_d) \subseteq I(if_d, OR)$ which together with the above containments yields $I(if_d, OR) = I(IF_d) = I(if_d, \Gamma_d)$. At last we prove that if D is finite then $if_d \in I(OR)$ and $OR \in I(\Gamma_d)$. Then we obtain the desired equalities $I(OR) = I(IF_d) = I(\Gamma_d)$.

Consider the strict function $g(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ of the type $b \to d$ defined as follows: $g(t, f, f, f) = 0$, $g(f, t, f, f) = 1$, $g(f, f, t, f) = 0$, $g(f, f, f, t) = 1$; in other cases g is undefined. Then, clearly,

$$if_d\ \alpha\ then\ x\ else\ y \equiv g(\alpha\ AND\ x = 0, \alpha\ AND\ x = 1, \neg\alpha\ AND\ y = 0, \neg\alpha\ AND\ y = 1) \tag{6}$$

and hence $if_d \in I(OR)$. Finally, let f be as above and $f^{-1}(\alpha)$ be the "decoding" function:

$$f^{-1}(\alpha) \equiv \begin{cases} 1 & \text{if } \alpha = true, \\ 0 & \text{if } \alpha = false, \\ \omega & \text{otherwise.} \end{cases}$$

Then $\alpha\, OR\, \beta \equiv f(\Gamma_d(1, f^{-1}(\alpha), f^{-1}(\beta))$ and hence $OR \in I(\Gamma_d)$ as desired.

(b) We now prove that if D is infinite then $\Gamma_d \notin I(IF_d)$. Suppose that it is not the case. Then there exists a canonical term τ in $I'(IF_d)$ representing Γ_d (cf. Lemma 1). Since Γ_d depends only on data variables x, y, z, tests and terminals in τ are either constants or strict functions of the types $d \to b$ and $d \to d$, respectively, depending on (some of) these variables.

Since D is infinite, there exists a constant $c_0 \in D$, $c_0 \neq \omega$, which does not occur among terminals of τ. Consider now three interpretations:

$$J_1\colon\ x \equiv y \equiv c_0,\ z \equiv \omega,$$

$$J_2\colon\ x \equiv z \equiv c_0,\ y \equiv \omega,$$

$$J_3\colon\ y \equiv z \equiv c_0,\ x \equiv \omega.$$

Let us construct a path in the corresponding termal tree Q as follows:

(a) the root of Q is at the beginning of the path;

(b) if a test already included in the path is either the constant ω or depends on all variables x, y, z, then either of its successors is included in the path;

(c) if this test does not depend on the variable x (y, z) then the successor is included which would be chosen under interpretation J_3 (J_2, J_1).

It is easy to see that this path belongs to each of the subtrees Q_{J_1}, Q_{J_2}, Q_{J_3}. Since $\Gamma_d(x, y, z)$ equals c_0 under all the three interpretations, the terminal of the path must yield the same value. This terminal cannot be a constant because of the choice of c_0; hence, it is a strict function. But then it yields the value ω (not c_0) under at least one of the three interpretations. This is a contradiction.

(c) Finally we prove that if D is infinite then $IF_d \notin I(if_d, OR)$. In fact it is necessary to prove that $IF_d \notin I(if_d, IF_b)$, since

α *OR* $\beta \equiv IF_b\ \alpha$ *then true else* β.

Suppose that $IF_d(\alpha, x, y) \in \mathrm{I}(if_d, IF_b)$. Then there exists a canonical term τ in $\mathrm{I}'(if_d, IF_b)$ representing IF_d (see Lemma 2). It follows from the proof of Lemma 2 that every initial b-function occurring in τ is either the constant ω or the variable α or a strict function of the type $d \to b$ depending on x, y, while every initial d-function is either a constant or a strict function of the type $d \to d$ depending on x, y.

Now choose c_0 as above and consider three interpretations:

J_1: $\alpha \equiv true$, $x \equiv c_0$, $y \equiv \omega$,

J_2: $\alpha \equiv false$, $x \equiv \omega$, $y \equiv c_0$,

J_3: $\alpha \equiv \omega$, $x \equiv c_0$, $y \equiv c_0$.

Let Q be the terminal tree corresponding to the first (in the order of execution) macrotest of τ; we construct a path in Q as follows:

(a) the root of Q is at the beginning of the path;

(b) if a test already included in the path is the constant ω, then either of its successors is included in the path;

(c) if this test is the variable α, then its *true*-successor is included in the path;

(d) if this test depends on one or both variables x, y then its successor is included which would be chosen under J_3.

Clearly, this path belongs to both Q_{J_1} and Q_{J_3}. Since $IF_d(\alpha, x, y)$ equals $c_0 \neq \omega$ under both J_1 and J_3, the values of all terminals in each Q_{J_1} and Q_{J_3} must be unequal to ω and must coincide (otherwise the value of the considered macrotest, and hence the value of τ, will become equal to ω). Since Q_{J_1} and Q_{J_3} have a common terminal, the macrotest yields the same value *true* or *false* under both J_1 and J_3. It can be shown by constructing a path belonging to both Q_{J_2} and Q_{J_3}, that the macrotest yields this value under J_2, as well.

Repeating the reasoning, we make sure that under all the three interpretations we pass through the same macrotests of τ and hence reach the same terminal. Its value, by our hypothesis, must be equal to c_0 in all the three cases. Since this terminal cannot be a constant because of the choice of c_0, it is a strict function. But then under at least one of J_1 and J_2 its value is ω, but not c_0. This is a contradiction.

This completes the proof of Theorem 1. □

Proof of Theorem 2. Replacing in (5) *OR* by *or* and IF_d by if_d yields $\mathrm{I}(or) \subseteq \mathrm{I}(if_d)$ for every D. If D is finite then replacing in (6) *AND* by *and* (here α *and* $\beta \equiv \neg(\neg\alpha\ or\ \neg\beta)$) yields the desired equality $\mathrm{I}(or) = \mathrm{I}(if_d)$. The proper containment $\mathrm{I}(or) \subsetneq \mathrm{I}(if_d)$ for infinite D can be easily proved by contradiction, namely by induction on depth of a term which is assumed to represent if_d. Details are omitted. □

Proof of Theorem 4. ($\mathrm{CI}(if_d) \subsetneq \mathrm{CSM}$) Let $A \subset \{0,1,2,\ldots\}$ be a recursively enumerable (r.e.) but non-recursive set and let $\bar{A}$ be the complement of A. Consider the function f given by

$$f(x,y,z) = \begin{cases} 1 & \text{if } x \not\equiv \omega \ \& \ y \equiv z \not\equiv \omega \text{ or } x \in A \ \& \ y \equiv 1, \\ \omega & \text{otherwise.} \end{cases}$$

Clearly, f is computable and sequential. We now prove that $f \notin \mathrm{CI}(if_d)$.

Suppose that it is not the case. Then according to Lemma 1 and the remark following the proof of Lemma 2, there exists a canonical term τ in $\mathrm{CI}'(if_d)$ representing f. Note that τ contains no variables but x, y, z. Let Q be the corresponding termal tree.

Let $a \not\equiv \omega$ and $\bar{a} \not\equiv \omega$ be arbitrary elements of the sets A and $\bar{A}$, respectively. Consider now four interpretations:

$$J_1\colon\ x \equiv a,\ y \equiv 1,\ z \equiv \omega,$$

$$J_2\colon\ x \equiv a,\ y \equiv 1,\ z \equiv 1,$$

$$J_3\colon\ x \equiv \bar{a},\ y \equiv 1,\ z \equiv 1,$$

$$J_4\colon\ x \equiv \bar{a},\ y \equiv 1,\ z \equiv \omega.$$

The function f (and hence the term τ) yields the value 1 under J_1, J_2, J_3 and the value ω under J_4.

Note that neither tests nor terminal of Q_{J_1} can depend on z, since otherwise the value of τ would equal ω under J_1. But then, obviously, the paths Q_{J_1} and Q_{J_2} coincide.

On the other hand, the terminal of Q_{J_3} or at least one of its tests does depend on z. Indeed, otherwise Q_{J_3} would coincide with Q_{J_4} whose terminal however has the value ω, but not 1.

Thus, whatever x_0 is, it belongs to A if and only if given an interpretation J: $x \equiv x_0$, $y \equiv 1$, $z \equiv 1$, the corresponding path Q_J contains a test or a terminal depending on z. This can be known effectively, since all tests passed through during constructing Q_J are computable and their values must be unequal to ω under J (since $f(x_0,1,1) \equiv 1$). Hence, A is recursive, which is a contradiction. □

8.

Before presenting the notions and lemmas necessary for Theorems 3 to 5 to be proved, we make the following remarks.

First, we will further consider monotonic d-functions only. For b-functions all the proofs are almost the same. One should only use if_b, IF_b and Γ_b for b-functions whenever if_d, IF_d and Γ_d are used for d-functions. Since Theorems 3 to 5 require,

however, the use of if_d, IF_d and Γ_d, it is sufficient to apply the "encoding" described in the proof of Theorem 1.

Secondly, one may assume without loss of generality, that only d-functions of the type $d \to d$ are considered in Theorems 4 and 5 and Lemmas 6 to 9 which are necessary to prove them. Indeed, since the use of IF_d is allowed in these theorems (see also Theorem 1), such a reduction can be realized by *IF-decomposition with respect to a boolean argument*. Namely, for any monotonic function $g(\alpha, \bar{x})$ (α is a boolean argument),

$$g(\alpha, \bar{x}) \equiv IF\ \alpha\ then\ g(true, \bar{x})\ else\ (IF\ \alpha\ then\ g(\omega, \bar{x})\ else\ g(false, \bar{x}))$$

holds (cf. (4)).

Taking these remarks into account we will everywhere omit the index and write *if*, *IF* and Γ for if_d, IF_d and Γ_d, respectively.

Let now $f(x_1, \ldots, x_n)$ be a monotonic function. To each nonempty subset of the argument set $\{x_1, \ldots, x_n\}$ there corresponds a strict *subfunction* of f. For example, the subfunction $f\langle x, y\rangle$ of $f(x, y, z)$ is defined as follows: If $x \not\equiv \omega$ and $y \not\equiv \omega$ then $f\langle x, y\rangle \equiv f(x, y, \omega)$, otherwise $f\langle x, y\rangle \equiv \omega$. Other subfunctions are similarly defined. Clearly, each subfunction of a computable function is also computable.

Nonempty pairwise non-intersecting sets of arguments are further denoted by Greek letters ξ, η, ζ (possibly with indices).

We now define the *union operation* $\cup$ on the set M, which is applied to functions depending, in general, on different arguments. Let, for example, $f_1(\xi, \eta)$ and $f_2(\xi, \zeta)$ be two monotonic functions. A function of arguments ξ, η, ζ denoted by $f_1(\xi, \eta) \cup f_2(\xi, \zeta)$ is the union of f_1 and f_2 if

(a) $\forall \xi, \eta, \zeta, [f_1(\xi, \eta) \cup f_2(\xi, \zeta) \supseteq f_1(\xi, \eta)]$,

(b) $\forall \xi, \eta, \zeta, [f_1(\xi, \eta) \cup f_2(\xi, \zeta) \supseteq f_2(\xi, \zeta)]$,

(c) for any function $U(\xi, \eta, \zeta)$ which satisfies (a) and (b),

$$U(\xi, \eta, \zeta) \supseteq f_1(\xi, \eta) \cup f_2(\xi, \zeta).$$

Clearly, the union exists only for f_1 and f_2 satisfying the following *compatability* condition:

$$\forall \xi, \eta, \zeta, \quad [f_1(\xi, \eta) \not\equiv \omega\ \&\ f_2(\xi, \zeta) \not\equiv \omega \to f_1(\xi, \eta) = f_2(\xi, \zeta)].$$

For example, any two subfunctions of a monotonic function are compatible. The union of a larger number of functions is similarly defined. For example, if $f_1(\xi, \eta)$, $f_2(\xi, \zeta)$ and $f_3(\xi)$ are pairwise compatible then

$$f_1(\xi, \eta) \cup f_2(\xi, \zeta) \cup f_3(\xi) \equiv (f_1(\xi, \eta) \cup f_2(\xi, \zeta)) \cup f_3(\xi).$$

The union of compatible monotonic functions is, obviously, monotonic. Lemmas 3 to 5 below follow immediately from the definitions.

Lemma 3. *Every monotonic function coincides with the union of all of its subfunctions.*

(In fact, it is sufficient here to consider subfunctions which do not equal ω identically).

Lemma 4. *Let functions f_1, f_2, f_3 and g be such that g is compatible with each f_i and $f_1 \equiv IF\ \alpha$ then f_2 else f_3. Then*

$$f_1 \cup g \equiv IF\ \alpha\ then\ f_2 \cup g\ else\ f_3 \cup g.$$

From this point on, when studying properties of compatible functions f_1, f_2 and constructing the union of such functions we will distinguish three possible correlations between their argument sets:

(a) *non-intersection*, e.g. $f_1(\xi)$ and $f_2(\eta)$;
(b) *enclosure*, e.g. $f_1(\xi)$ and $f_2(\xi, \eta)$;
(c) *engagement*, e.g. $f_1(\xi, \eta)$ and $f_2(\xi, \zeta)$.

The first case is considered in Lemma 5, the second in Lemmas 6 and 7 and the third in Lemmas 5, 8 and 9.

Lemma 5. (1) *Let $f_1(\xi)$ and $f_2(\eta)$ be two compatible functions which do not equal ω identically. Then*

$$\exists c \neq \omega, \forall \xi, \eta, \ \{[f_1(\xi) \neq \omega \to f_1(\xi) = c] \,\&\, [f_2(\eta) \neq \omega \to f_2(\eta) = c]\}.$$

(2) *Let $f_1(\xi, \eta)$ and $f_2(\xi, \zeta)$ be compatible and let ξ_0, η_0, ζ_0 be such that $f_1(\xi_0, \eta_0) \neq \omega$ and $f_2(\xi_0, \zeta_0) \neq \omega$.*

Then

$$\exists c(\xi_0), \forall \eta, \zeta, \quad \{[f_1(\xi_0, \eta) \neq \omega \to f_1(\xi_0, \eta) = c(\xi_0)] \,\&\, [f_2(\xi_0, \zeta) \neq \omega \to f_2(\xi_0, \zeta) = c(\xi_0)]\}.$$

A function $f(\xi, \eta)$ is *ξ-strict* if $f(\xi, \eta) \equiv \omega$ whenever at least one argument from the argument subset ξ equals ω. In particular, if a function $f(\xi)$ is ξ-strict then it is strict in the usual sense.

Lemma 6. *Let $f_1(\xi)$ and $f_2(\xi, \eta)$ be compatible ξ-strict functions. Then there exist a strict predicate $p(\xi)$ and a strict function $\varphi(\xi)$ such that*

$$f_1(\xi) \cup f_2(\xi, \eta) \equiv IF\ p(\xi)\ then\ \varphi(\xi)\ else\ f_2(\xi, \eta). \tag{7}$$

If f_1 and f_2 are computable then p and φ are also computable.

Proof. Consider three sets:

$$S_1 = \{\xi \mid f_1(\xi) \not\equiv \omega\}, \qquad S_2 = \{\xi \mid \exists \eta,\ f_2(\xi, \eta) \not\equiv \omega\},$$

$$S_3 = \{\xi \mid \exists \eta_1, \eta_2,\ \omega \not\equiv f_2(\xi, \eta_1) \not\equiv f_2(\xi, \eta_2) \not\equiv \omega\}.$$

Since f_1 and f_2 are compatible, $S_1 \cap S_3 = \emptyset$. Therefore the strict predicate can be defined by:

$$p(\xi) \equiv \begin{cases} \textit{true} & \text{if } \xi \in S_1, \\ \textit{false} & \text{if } \xi \in S_3, \\ \omega & \text{otherwise.} \end{cases}$$

Consider further a strict function $\varphi(\xi)$ satisfying the following conditions:

(a) $\varphi(\xi) \not\equiv \omega \leftrightarrow \xi \in S_1 \cup S_2$,

(b) $\varphi(\xi) = t \not\equiv \omega \leftrightarrow f_1(\xi) = t \vee \exists \eta [f_2(\xi, \eta) = t]$.

It is easy to see that with p and φ chosen in such a way (7) is really fulfilled.

If now f_1 and f_2 are computable then S_1 and S_3 are r.e. sets and hence $p(\xi)$ is computable. Finally we show how $\varphi(\xi)$ can be computed. Let $\mathfrak{M}$ and $\mathfrak{N}$ be algorithms computing f_1 and f_2, respectively. Let $\mathfrak{M}$ and $\mathfrak{N}$ run simultaneously as follows. $\mathfrak{M}$ is applied to the arguments ξ, while $\mathfrak{N}$ makes one step on the arguments $\xi, 0, \ldots, 0$, then two steps on the arguments $\xi, 0, \ldots, 0$ and $\xi, 0, \ldots, 1$, etc. If one of the algorithms stops first with a result t, then $\varphi(\xi) \equiv t$; otherwise $\varphi(\xi) \equiv \omega$. □

A sequence of subfunctions of a function f having the form $f\langle\xi_1\rangle$, $f\langle\xi_1, \xi_2\rangle, \ldots, f\langle\xi_1, \xi_2, \ldots, \xi_\nu\rangle$ is called a *P-chain of f*.

Lemma 7. *Let* $f \in \mathrm{M}$ ($f \in \mathrm{CM}$). *Then for every P-chain of f,*

$$f\langle\xi_1\rangle \cup f\langle\xi_1, \xi_2\rangle \cup \ldots \cup f\langle\xi_1, \xi_2, \ldots, \xi_\nu\rangle \in \mathrm{I}(IF) \quad (\mathrm{CI}(IF)).$$

Proof. For $\nu = 2$ this follows immediately from Lemma 6. The same lemma also allows us to increase length of a chain. To show this it is sufficient to note that the union of ξ-strict functions is itself ξ-strict. For example, if $\nu = 3$ then applying equality (7) first for $f\langle\xi_1\rangle$ and $f\langle\xi_1, \xi_2\rangle \cup f\langle\xi_1, \xi_2, \xi_3\rangle$ and then for $f\langle\xi_1, \xi_2\rangle$ and $f\langle\xi_1, \xi_2, \xi_3\rangle$ we obtain the desired representation of $f\langle\xi_1\rangle \cup f\langle\xi_1, \xi_2\rangle \cup f\langle\xi_1, \xi_2, \xi_3\rangle$. □

Lemma 8. *Let* $f_1(\xi, \eta)$ *and* $f_2(\xi, \zeta)$ *be compatible* ξ*-strict functions. Then there exist a strict predicate* $q(\xi)$ *and a strict function* $c(\xi)$ *such that*

$$c(\xi) \cup f_1(\xi, \eta) \cup f_2(\xi, \zeta) \equiv \tag{8}$$

$$\equiv IF\ q(\xi)\ then\ c(\xi) \cup f_1(\xi, \eta)\ else\ c(\xi) \cup f_2(\xi, \zeta).$$

If f_1 *and* f_2 *are computable then* q *and* c *are also computable.*

Proof. Consider three sets

$$R_1 = \{\xi \mid \exists \eta,\ f_1(\xi, \eta) \not\equiv \omega\}, \qquad R_2 = \{\xi \mid \exists \zeta,\ f_2(\xi, \zeta) \not\equiv \omega\},$$

$$R = R_1 \cap R_2.$$

Since f_1 and f_2 are compatible, then by Lemma 5 (2) there exists a strict function $c(\xi)$ such that

(a) $c(\xi) \not\equiv \omega \leftrightarrow \xi \in R$,
(b) $\xi \in R \ \&\ f_1(\xi, \eta) \not\equiv \omega \rightarrow f_1(\xi, \eta) = c(\xi)$,
(c) $\xi \in R \ \&\ f_2(\xi, \zeta) \not\equiv \omega \rightarrow f_2(\xi, \zeta) = c(\xi)$.

Consider further two sets A and B such that:

$$A \supseteq R_1 \setminus R_2; \qquad B \supseteq R_2 \setminus R_1; \qquad A \cup B = R_1 \cup R_2; \qquad A \cap B = \emptyset$$

(in other words A and B separate the sets $R_1 \setminus R_2$ and $R_2 \setminus R_1$). Define now the strict predicate

$$q(\xi) \equiv \begin{cases} \textit{true} & \text{if } \xi \in A, \\ \textit{false} & \text{if } \xi \in B, \\ \omega & \text{otherwise.} \end{cases}$$

It is easy to verify that with q and c chosen in such a way that (8) is really fulfilled.

If now f_1 and f_2 are computable then R_1, R_2 and R are r.e. sets. Taking in this case r.e. separating sets A and B (it is easy to understand that such ones exist) we obtain the computable predicate $q(\xi)$. Finally we show how $c(\xi)$ can be computed. Let $\mathfrak{M}$ and $\mathfrak{N}$ be algorithms computing f_1 and f_2 respectively. Let them run simultaneously as follows. First each of them makes one step on the arguments $\xi, 0, \ldots, 0$, then two steps on the arguments $\xi, 0, \ldots, 0$ and $\xi, 0, \ldots, 1$, etc. If $\xi \in R$ then a moment will come such that both $\mathfrak{M}$ and $\mathfrak{N}$ stop on some sets of arguments. In this case both $\mathfrak{M}$ and $\mathfrak{N}$ yield the same result t (due to compatibility of f_1 and f_2); then also $c(\xi) = t$. Otherwise at least one of two algorithms runs infinitely, and $c(\xi) \equiv \omega$. □

Let $f\langle\xi\rangle$, $f\langle\xi, \eta\rangle$ and $f\langle\xi, \zeta\rangle$ be subfunctions of a function f. We introduce the following denotation: $f\langle\xi, \eta\rangle \perp_{\mathscr{C}} f\langle\xi, \zeta\rangle$ if $f\langle\xi\rangle \subseteq \mathscr{C} \subseteq f$.

Lemma 9. *Let* $f \in \mathrm{SM}$ *and* $f\langle\xi, \eta\rangle \perp_{\mathscr{C}} f\langle\xi, \zeta\rangle$. *Then*

$$\mathscr{C} \cup f\langle\xi, \eta\rangle \cup f\langle\xi, \zeta\rangle \equiv \tag{9}$$

$$\equiv \textit{IF } q(\xi) \textit{ then } \mathscr{C} \cup f\langle\xi, \eta\rangle \textit{ else } \mathscr{C} \cup f\langle\xi, \zeta\rangle,$$

where $q(\xi)$ *is the same as in Lemma* 8.

Proof. It follows immediately from the definition of sequential functions that under conditions of the lemma the set R coincides with the set of those ξ, on which $f\langle\xi\rangle$

differs from ω. Moreover, in this case $c(\xi)$ coincides with $f\langle\xi\rangle$. Therefore equality (8) takes the form

$$f\langle\xi\rangle \cup f\langle\xi,\eta\rangle \cup f\langle\xi,\zeta\rangle \equiv$$
$$\equiv \mathit{IF}\ q(\xi)\ \mathit{then}\ f\langle\xi\rangle \cup f\langle\xi,\eta\rangle\ \mathit{else}\ f\langle\xi\rangle \cup f\langle\xi,\zeta\rangle.$$

Obviously, $\mathscr{C}$ is compatible with each of the considered subfunctions and hence it is compatible with the unions of these subfunctions. Therefore Lemma 4 can be applied:

$$\mathscr{C} \cup f\langle\xi\rangle \cup f\langle\xi,\eta\rangle \cup f\langle\xi,\zeta\rangle \equiv$$
$$\equiv \mathit{IF}\ q(\xi)\ \mathit{then}\ \mathscr{C} \cup f\langle\xi\rangle \cup f\langle\xi,\eta\rangle$$
$$\mathit{else}\ \mathscr{C} \cup f\langle\xi\rangle \cup f\langle\xi,\zeta\rangle.$$

Since $\mathscr{C} \cup f\langle\xi\rangle \equiv \mathscr{C}$, this yields the desired equality (9). □

9.

In this section we prove Theorems 3, 4 (the containment $\mathrm{CSM} \subsetneq \mathrm{CI}(\mathit{IF})$) and 5. All proofs are based on Lemma 3.

Proof of Theorem 3. The containment $\mathrm{I}(\mathit{if}) \subseteq \mathrm{SM}$ follows immediately from the closeness of the class SM with respect to composition [8]. The closeness holds also for the class ESM and therefore $\mathrm{CI}(\mathit{if}) \subseteq \mathrm{ESM}$ (every computable strict function is, obviously, effectively sequential).

To prove the inverse containment we note first that if x_i is a critical argument of a monotonic function f then all its subfunctions, which do not equal ω identically, depend on x_i. We assume that $f \in \mathrm{SM}$ is not a constant function (otherwise the theorem is proved); then such x_i necessarily exists (see point (1) of the definition of a strategy of sequential computation). Further for simplicity we consider three-ary function $f(x, y, z)$ only, with x playing the role of the mentioned x_i.

Note that instead of the system of functions $\{F_i\}$ which forms the strategy of sequential computation of f, the system of strict predicates $\{F_{ij}\}$ can be considered. For example, if the strategy is $\{F_1 = 1,\ F_2(x),\ F_3(x, y),\ F_4(x, z),\ F_5(x, y, z)\}$ then

$$F_{22}(x) \equiv \begin{cases} \mathit{true} & \text{if } F_2(x) = 2 \text{ (the number of the argument } y\text{)}, \\ \mathit{false} & \text{otherwise;} \end{cases}$$

$$F_{42}(x, z) \equiv \begin{cases} \mathit{true} & \text{if } F_4(x, z) = 2 \text{ (the number of the argument } y\text{)}, \\ \mathit{false} & \text{otherwise.} \end{cases}$$

Other predicates are similarly defined. Then

$$f(x, y, z) \equiv \textit{if } F_{22}(x) \textit{ then } (\textit{if } F_{33}(x, y) \textit{ then } f\langle x, y, z\rangle \textit{ else } f\langle x, y\rangle)$$
$$\textit{else } (\textit{if } F_{23}(x) \textit{ then } (\textit{if } F_{42}(x, z) \textit{ then } f\langle x, y, z\rangle$$
$$\textit{else } f\langle x, z\rangle)$$
$$\textit{else } f\langle x\rangle).$$

Since in the general case the value of F_i is unknown, one ought to insert a few tests of the form "$F_i = i$" which allows one to determine this value.

If now f is effectively sequential then all F_i and hence all F_{ij} are computable. Since all subfunctions of f are also computable, the right-hand side of the above equality belongs to CI(*if*). □

Proof of Theorem 4 (CSM$\subsetneq$CI(*IF*)). Given a function $f \in$ CM, we define the class $P(f)$ of terms and corresponding functions. A term τ belongs to $P(f)$ iff it differs from a canonical term in CI′(*IF*) only in that each of its terminals is not an initial function, but the union of all elements of some P-chain of f. It follows from Lemma 7 that

$$F \in P(f) \rightarrow F \in \mathrm{CI}(IF).$$

Let us extend $P(f)$ to the class $\tilde{P}(f) = \{F \mid \exists F',\ F' \in P(f)\ \&\ F \subseteq F'\}$. Obviously, $F \in \tilde{P}(f) \rightarrow F \subseteq f$.

Below we show for a non-constant function $f(x_1, \ldots, x_n) \in$ CSM that $f \in P(f)$ and hence $f \in \mathrm{CI}(IF)$. The proof proceeds by induction on n.

It should be reminded that all subfunctions of a sequential function which do not equal ω identically possess a common argument. Let x_1 be this argument.

The basis of induction is evident, since for $n = 1$, f is an initial function and for $n = 2$, $f(x_1, x_2) \equiv f\langle x_1\rangle \cup f\langle x_1, x_2\rangle$. Suppose now that the theorem is proved for n-ary functions ($n \geq 2$) and consider a function $f(x_1, \ldots, x_n, x_{n+1})$. Let us represent f as follows:

$$f(x_1, \ldots, x_n, x_{n+1}) \equiv f(x_1, \omega, x_3, \ldots, x_{n+1}) \cup \ldots \cup f(x_1, \ldots, x_{n-1}, \omega, x_{n+1})$$
$$\cup f(x_1, \ldots, x_n, \omega) \cup f\langle x_1, \ldots, x_n, x_{n+1}\rangle.$$

Let us denote the first n functions in the right-hand side of the equality by $\phi_1, \ldots, \phi_n$ and the last one by $\mathfrak{A}$ and prove that their union belongs to $P(f)$. For this we use the following scheme of induction:

Scheme 1. Let $\mathfrak{A}, \phi_1, \ldots, \phi_\nu$ be some functions and let a class K be such that

(1) $\forall i$, $[\mathfrak{A} \cup \phi_i \in K]$,

(2) $\forall i, j$, $[\mathfrak{L} \cup \phi_i \in K \ \ \& \ \ \mathfrak{L} \cup \phi_j \in K \rightarrow \mathfrak{L} \cup \phi_i \cup \phi_j \in K]$, where $\mathfrak{L} \equiv \mathfrak{A} \cup \phi_{i_1} \cup \ldots \cup \phi_{i_m}$.

Then $\mathfrak{A} \cup \phi_1 \cup \ldots \cup \phi_\nu \in K$.

Show that if $K = \tilde{P}(f)$ then conditions (1) and (2) of scheme 1 are satisfied, which implies $f \in \tilde{P}(f)$ and hence $f \in P(f)$, as desired.

(1) Note that each ϕ_i is an n-ary function from CSM, so that $\phi_i \in P(\phi_i)$ by induction hypothesis. Subfunctions of ϕ_i can naturally be considered as those of f and hence $\phi_i \in P(f)$. Let τ be a term in $P(f)$ representing ϕ_i. Then the corresponding term τ' for $\mathfrak{A} \cup \phi_i$ is obtained from τ by replacing each terminal in τ by the union of this terminal with the subfunction $\mathfrak{A}$ (repeated application of Lemma 4). Since $\mathfrak{A}$ depends on all the arguments of f, $\tau' \in P(f)$ and all the more $\tau' \in \tilde{P}(f)$.

(2) Let, for example, $\mathscr{L} \cup \phi_{n-1} \in \tilde{P}(f)$ and $\mathscr{L} \cup \phi_n \in \tilde{P}(f)$. Represent ϕ_{n-1} and ϕ_n as follows:

$$\phi_{n-1} \equiv f(x_1, \ldots, x_{n-1}, \omega, \omega) \cup \alpha_1 \cup \ldots \cup \alpha_\nu,$$

$$\phi_n \equiv f(x_1, \ldots, x_{n-1}, \omega, \omega) \cup \beta_1 \cup \ldots \cup \beta_\mu,$$

where $\alpha_1, \ldots, \alpha_\nu$ and $\beta_1, \ldots, \beta_\mu$ are subfunctions of f lacking for obtaining ϕ_{n-1} and ϕ_n, respectively. For example, for $n = 3$,

$$\phi_2 \equiv f(x_1, x_2, \omega, \omega) \cup f\langle x_1, x_4\rangle \cup f\langle x_1, x_2, x_4\rangle,$$

$$\phi_3 \equiv f(x_1, x_2, \omega, \omega) \cup f\langle x_1, x_3\rangle \cup f\langle x_1, x_2, x_3\rangle.$$

(We retain as α_i and β_j only those subfunctions which do not equal ω identically. Therefore those of them which are independent of x_1, are omitted.)

Denote $f(x_1, \ldots, x_{n-1}, \omega, \omega)$ by γ. The following statement holds true:

$$\forall i, j, [\alpha_i \perp_\gamma \beta_j].$$

Indeed, first, the arguments of α_i and β_j are engaged, since α_i depends at least on x_1 and x_{n+1} while β_j depends at least on x_1 and x_n. Secondly, γ depends on all arguments which are common for ϕ_{n-1} and ϕ_n and all the more on all arguments $x_{m_1}, \ldots, x_{m_s}$ which are common for α_i and β_j. Hence,

$$f\langle x_{m_1}, \ldots, x_{m_s}\rangle \subseteq \gamma \subseteq f$$

as desired.

Now make use of one more scheme of induction:

Scheme 2. Let $\varphi, \alpha_1, \ldots, \alpha_\nu, \beta_1, \ldots, \beta_\mu$ be some functions and let a class K be such that

(1) $\forall i, [\varphi \cup \alpha_1 \cup \ldots \cup \alpha_i \in K]$, $\forall j, [\varphi \cup \beta_1 \cup \ldots \cup \beta_j \in K]$,

(2) $\forall i, j, [\mathscr{C} \cup \alpha_i \in K \;\&\; \mathscr{C} \cup \beta_j \in K \rightarrow \mathscr{C} \cup \alpha_i \cup \beta_j \in K]$,

where $\mathscr{C} \equiv \varphi \cup \alpha_1 \cup \ldots \cup \alpha_{i-1} \cup \beta_1 \cup \ldots \cup \beta_{j-1}$.

Then $\varphi \cup \alpha_1 \cup \ldots \cup \alpha_\nu \cup \beta_1 \cup \ldots \cup \beta_\mu \in K$.

Indeed, by condition (1) of this scheme one has $\varphi \cup \alpha_1 \in K$ and $\varphi \cup \beta_1 \in K$ and hence by condition (2), $\varphi \cup \alpha_1 \cup \beta_1 \in K$ which together with $\varphi \cup \alpha_1 \cup \alpha_2 \in K$ yields $\varphi \cup \alpha_1 \cup \alpha_2 \cup \beta_1 \in K$, etc.

Show that if $K = \tilde{P}(f)$ and $\varphi \equiv \mathscr{L} \cup \gamma$ then conditions (1) and (2) of scheme 2 are satisfied. This will imply $\mathscr{L} \cup \phi_{n-1} \cup \phi_n \in \tilde{P}(f)$, as desired.

Fulfillment of condition (1) follows immediately from $\mathscr{L} \cup \phi_{n-1} \in \tilde{P}(f)$ and

$\mathfrak{L} \cup \phi_n \in \tilde{P}(f)$. As for condition (2), note first that $\alpha_i \perp_\gamma \beta_j$ implies $\alpha_i \perp_{\mathfrak{C}} \beta_j$. Then it follows from equality (9) in Lemma 9 and from the containments $\mathfrak{C} \cup \alpha_i \in \tilde{P}(f)$ and $\mathfrak{C} \cup \beta_j \in \tilde{P}(f)$ that $\mathfrak{C} \cup \alpha_i \cup \beta_j \in \tilde{P}(f)$.

This completes the proof of Theorem 4. □

Proof of Theorem 5. Consider the following scheme of induction:

Scheme 3. Let $\phi_1, \ldots, \phi_\nu$ be some function and let a class K be such that

(1) $\forall i, j, [\phi_i \cup \phi_j \in K]$,

(2) $\forall i, j, m, [\mathfrak{A} \cup \phi_i \cup \phi_j \in K \;\&\; \mathfrak{A} \cup \phi_i \cup \phi_m \in K \;\&\; \mathfrak{A} \cup \phi_j \cup \phi_m \in K \rightarrow \mathfrak{A} \cup \phi_i \cup \phi_j \cup \phi_m \in K]$, where $\mathfrak{A} \equiv \phi_{i_1} \cup \ldots \cup \phi_{i_s}$.

Then $\phi_1 \cup \ldots \cup \phi_\nu \in K$.

Let now $f(x_1, \ldots, x_n) \in \mathrm{M(CM)}$ and let $\{\phi_i\}$ be the set of all subfunctions of f. We show that if $K = \mathrm{I}(if, \Gamma)$ ($\mathrm{CI}(if, \Gamma)$) then conditions (1) and (2) of scheme 3 are satisfied. By Lemma 3, this will imply the fulfillment of Theorem 5.

Condition (2) is satisfied, since

$$\mathfrak{A} \cup \phi_i \cup \phi_j \cup \phi_m \equiv \Gamma(\mathfrak{A} \cup \phi_i \cup \phi_j, \mathfrak{A} \cup \phi_i \cup \phi_m, \mathfrak{A} \cup \phi_j \cup \phi_m).$$

Thus, it is sufficient to verify that condition (1) is satisfied. Anticipating a little we should like to point out that the union of any two subfunctions of f can be expressed without applying Γ by using the less powerful (see Theorem 1) functions *IF* and *OR*.

Consider now all the three correlations between the argument sets of two subfunctions.

(1) *Non-intersection*: $\phi_i(\xi)$, $\phi_j(\eta)$. Define two strict predicates

$$p(\xi) \equiv \begin{cases} true & \text{if } \phi_i(\xi) \not\equiv \omega, \\ \omega & \text{otherwise;} \end{cases}$$

$$q(\eta) \equiv \begin{cases} true & \text{if } \phi_j(\eta) \not\equiv \omega. \\ \omega & \text{otherwise.} \end{cases}$$

Let c be the constant provided by point (1) of Lemma 5. Then

$$\phi_i(\xi) \cup \phi_j(\eta) \equiv IF\ p(\xi)\ then\ c\ else\ (if\ q(\eta)\ then\ c\ else\ \omega).$$

Clearly, if f is computable then so are p and q.

(2) *Enclosure*: $\phi_i(\xi)$, $\phi_j(\xi, \eta)$. See Lemma 7.

(3) *Engagement*: $\phi_i(\xi, \eta)$, $\phi_j(\xi, \zeta)$. It follows from equality (8) in Lemma 8 that the function G given by

$$G(\xi, \eta, \zeta) \equiv c(\xi) \cup \phi_i(\xi, \eta) \cup \phi_j(\xi, \zeta)$$

belongs to $\mathrm{I}(IF)$ (if f is computable then G belongs to $\mathrm{CI}(IF)$). Obviously, $G \supseteq \phi_i \cup \phi_j$ and the proper inequality $\supset$ holds only for such ξ_0, η_0, ζ_0 that $c(\xi_0) \not\equiv \omega$ but $\phi_i(\xi_0, \eta_0) \equiv \omega$ and $\phi_j(\xi_0, \zeta_0) \equiv \omega$. Therefore

$$\phi_i(\xi, \eta) \cup \phi_j(\xi, \zeta) \equiv \textit{if } \mathrm{def}(\phi_i(\xi, \eta)) \textit{ OR } \mathrm{def}(\phi_j(\xi, \zeta))$$
$$\textit{then } G(\xi, \eta, \zeta) \textit{ else } \omega,$$

where

$$\mathrm{def}(x) \equiv \begin{cases} \textit{true} & \text{if } x \neq \omega, \\ \omega & \text{otherwise.} \end{cases}$$

This completes the proof of Theorem 5. □

References

[1] Z. Manna, S Ness and J. Vuillemin, Inductive methods for proving properties of programs, *Comm. ACM* **16** (8) (1973) 491–502.

[2] J. McCarthy, A basis for a mathematical theory of computation, in: *Computer Programming and Formal Systems*, P. Braffort and D. Hirschberg (eds.) (Humanities Press, New York, 1963) 33–70.

[3] M. Paterson and C. Hewitt, Comparative schematology, in: *Record of Project MAC Conference on Concurrent Systems and Parallel Computation* (Association for Computing Machinery, New York, 1970) 119–128.

[4] V. Yu. Sazonov, Sequentially and parallely computable functionals (extended abstract), in: *Proc. Symp. on λ-Calculus and Computer Science Theory*, Rome, Lecture Notes in Computer Science **37** (Springer, Berlin, 1975) 312–318.

[5] D. Scott, A type-theoretical alternative to CUCH, ISWIM, OWHY, Oxford (1969).

[6] D. Scott, Outline of a mathematical theory of computation, Oxford University Computing Lab, PRG–2, Oxford (1970).

[7] M. B. Trakhtenbrot, On interpreted functions in program schemata, in: *System and Theoretical Programming*, V. E. Kotov (ed.), Novosibirsk (1973) 188–211 (in Russian).

[8] J. Vuillemin, Proof techniques for recursive programs, Ph. D. Thesis, Computer Science Dept., Stanford University, Stanford (1973).